Electronic Notes in Theoretical Computer Science 9 (1997)
URL: http://www.elsevier.nl/locate/entcs/volume9.html 13 pages

Verifying Invariants by Approximate Image Computation

Felice Balarin

Cadence Berkeley Laboratories
Berkeley, USA

Abstract

Automatic formal verification of safety properties typically requires computing reachable states of a system. A more efficient (and less automatic) alternative is to check whether a user suggested superset of reachable states is an invariant, i.e. whether it contains its image specified by the transition relation of the system. Still, this approach may be prohibitively expensive due to the complexity of image computation. To alleviate this problem we suggest to use approximate image computations, and we show that even though the approximation computes a superset of the image, it can, in certain cases, be used to answer categorically the question whether the suggested invariant contains its image. More precisely, we first establish sufficient conditions that the approximate image computation and the suggested invariant need to satisfy in order to always reach a conclusive result of the verification process. Then, we use these results to show that the three approximate image computation methods proposed previously for approximate reachability analysis could be used for exact invariant verification.

1 Introduction

Many systems, from computer programs and communication protocols to industrial processes and embedded controllers, can be specified by some set of *states* S and a *transition relation* $T \subseteq S \times S$ which contains all pairs of states (s, q) such that the system can move from s to q "in one step". There are many formalisms that fit this broad description (e.g. L-processes [Kur94], synchronous programs [BC84], finite-state [HU79], timed [AD90], and hybrid [ACHH93] automata, ...), all of which differ in the possible sets of states, and the precise meaning of the phrase "in one step".

Common to all these formalisms is that proving many of their properties reduces to verifying that, starting from any of the designated *initial* states, the system never leaves the set of *acceptable* states. For example, acceptable states of a traffic light controller might be those in which pedestrian and car signals are not open at the same time, and acceptable states of a cache consistency protocol might be those in which all the copies of a data item are consistent. Such properties are often referred to as *safety* properties.

Safety properties are often verified by first computing the set of *reachable* states of the system, and then checking that the reachable states are all acceptable. Computing the reachable states is typically done by:

(i) setting the current set of states to be the set initial states,

(ii) adding to current set of states its *image*. The image $\mathcal{T}(Q)$ of a set of states $Q \subseteq S$ is defined by:

$$\mathcal{T}(Q) \;=\; \{s \mid \exists q \in Q : (q,s) \in T\} \;.$$

Intuitively, it represents the set of states reachable from Q in one step.

(iii) repeating step 2 until convergence.

If the iteration terminates, then the final current set of states represents exactly the set of states reachable from some of the initial states.

The problem with this approach is that if S is not finite, then the iteration might not terminate. Moreover, even if the termination is assured, the number of iterations might be prohibitively large. One approach to this problem is to approximate the image of a set with its superset. A well chosen approximation might be easier to compute, or it might improve the convergence. The final result of such a computation is a superset of reachable states. If that superset is contained in the set of acceptable states, then the system is correct. Otherwise, the outcome of the verification process is inconclusive.

Another (less automatic, but also computationally cheaper) approach to verification of safety properties is to let the user suggest an *invariant*. An invariant is any superset I of initial states that satisfies:[1]

$$\mathcal{T}(I) \;\subseteq\; I \;. \tag{1}$$

If we can show that the user suggested set is indeed an invariant, and that it is contained in the set of acceptable states, then the system would be verified.

Verifying safety properties using invariants still involves a single image computation. In certain cases, this can be a complex operation involving a symbolic execution of a program, or a solution to a set of differential equations. Thus, it would be desirable to be able to check (1) using cheaper but approximate image computation, rather than the exact one. More precisely, we seek an approximate image operator $\widehat{\mathcal{T}}$ such that:

$$\mathcal{T}(I) \;\subseteq\; I \quad \text{iff} \quad \widehat{\mathcal{T}}(I) \;\subseteq\; I \;. \tag{2}$$

The main result of this paper is that the three approximate image operators suggested earlier for approximate reachability analysis of synchronous programs [Hal93] and timed automata [Bal96] satisfy (2), provided that the suggested invariant satisfies some mild conditions. In all of the three cases the approximate operators are such that $\mathcal{T}(Q) \subseteq \widehat{\mathcal{T}}(Q)$ holds for any set of states Q. Thus, proving one direction of (2) is easy. Proving the other direction is tricky, and obviously does not hold for arbitrary I (except in the trivial case $\widehat{\mathcal{T}} \equiv \mathcal{T}$).

[1] There is no universally accepted definition of an invariant. Sometimes, an invariant is defined to be any superset of reachable states, while any invariant satisfying (1) is called an *inductive invariant*.

In the rest of this paper, we first examine in Section 2 rather general conditions under which (2) holds. Then, in Section 3 we apply these results to approximate image computation methods suggested in [Hal93] for analysis of automata produced by compilers of synchronous languages. In Section 4 we apply the general results to two approximate image computation methods suggested in [Bal96] for analysis of timed automata.

2 General Framework

We consider some set (of states) S and some class of its subsets $\mathcal{C} \subseteq 2^S$. Given some set $Q \subseteq S$, we say that some $x \in \mathcal{C}$ is an *approximation* of Q, if x contains Q. We say that x is the *best approximation* of Q in $\mathcal{C}$ (and write $x = \lceil Q \rceil^{\mathcal{C}}$), if x is an approximation of Q and it is contained in all other approximations.

The best approximation may not always exist. The following proposition gives conditions on $\mathcal{C}$ for its existence.

Proposition 2.1 *Every $Q \subseteq S$ has a unique best approximation in $\mathcal{C}$ if and only if $\mathcal{C}$ is such that:*

- $S \in \mathcal{C}$,
- $\mathcal{C}$ *is closed under intersection, i.e.:*

$$\forall \mathcal{D} \subseteq \mathcal{C} : \{s \in S \,|\, \forall x \in \mathcal{D} : s \in x\} \in \mathcal{C} \ . \tag{3}$$

If $\mathcal{C}$ is finite, then condition (3) simplifies to $\mathcal{C}$ being closed under pairwise intersection (i.e. for every $x, y \in \mathcal{C}$, it must be that $x \cap y$ is also in $\mathcal{C}$). However, if $\mathcal{C}$ is infinite, (3) may not hold even if $\mathcal{C}$ is closed under pairwise intersection. For example, the set convex polyhedra is closed under the pairwise but not under infinite intersection, and in fact not every set of points (e.g. a circle in a plane) has the best approximation by convex polyhedra. Fortunately, to compute an image approximation, it is not necessary for every set of states to have the best approximation. The approximate image is usually computed by applying a sequence of operations to elements of $\mathcal{C}$. It suffices to show that the results of applying these operations to elements of $\mathcal{C}$ always have the best approximation.

The following simple observation provides a justification for using approximate image computation for invariant verification:

Proposition 2.2 *For every $x \in \mathcal{C}$ and every $Q \subseteq S$ such that $\lceil Q \rceil^{\mathcal{C}}$ exists:*

$$Q \subseteq x \quad \textit{iff} \quad \lceil Q \rceil^{\mathcal{C}} \subseteq x$$

In particular, to check whether some $x \in \mathcal{C}$ is an invariant, it is both necessary and sufficient that it contains its approximate image. However, this can be beneficial only if computing an approximate image is cheaper than computing the exact one. Images are often computed by a sequence of operations on subsets of states. Let $f : 2^S \mapsto 2^S$ be such an operator.[2] We

[2] In this section, we will present all results for unary operators only. The extension to n-ary operators is straightforward, but notationally cumbersome.

would would like to replace f with its approximation $f^{\mathcal{C}} : \mathcal{C} \mapsto \mathcal{C}$, defined simply by:

$$f^{\mathcal{C}}(x) \;=\; \lceil f(x) \rceil^{\mathcal{C}} \quad \text{for every} \quad x \in \mathcal{C} \;\; .$$

In the rest of this section we show that if operators have certain properties, then the best image approximation can be computed by replacing each operator with its approximation.

The key property of operators that enable our approach is the ability to propagate approximations, i.e. to be able to approximate arguments if only an approximate result is needed, and vice versa. More precisely we say that f *in-propagates* $\mathcal{C}$ if:

$$\lceil f(x) \rceil^{\mathcal{C}} \;=\; f^{\mathcal{C}}(\lceil x \rceil^{\mathcal{C}}) \quad \text{for every} \quad x \in 2^S \;\; .$$

Similarly, we say that f *out-propagates* $\mathcal{C}$ if:

$$f(\lceil x \rceil^{\mathcal{C}}) \;=\; f^{\mathcal{C}}(\lceil x \rceil^{\mathcal{C}}) \quad \text{for every} \quad x \in 2^S \;\; .$$

Operators that out-propagate $\mathcal{C}$ have the following simple characterization:

Proposition 2.3 *An operator f out-propagates $\mathcal{C}$ if and only if $\mathcal{C}$ is closed with respect to f.*

It is not known at the moment whether in-propagating operators have a similar characterization.

An approximate composition of in- or out-propagating functions can computed by composing approximate operators. This follows from the following series of propositions, in which we use $f \circ g$ to denote the composition of f and g defined by $(f \circ g)(x) = f(g(x))$.

Proposition 2.4 *If operators f and g in-propagate $\mathcal{C}$, then so does also their composition $f \circ g$, and in addition:*

$$(f \circ g)^{\mathcal{C}} \;\equiv\; f^{\mathcal{C}} \circ g^{\mathcal{C}} \;\; . \tag{4}$$

Proposition 2.5 *If operators f and g out-propagate $\mathcal{C}$, then so does also their composition $f \circ g$, and in addition:*

$$(f \circ g)^{\mathcal{C}} \;\equiv\; f^{\mathcal{C}} \circ g^{\mathcal{C}} \;\; . \tag{5}$$

Proposition 2.6 *If operator f in-propagates $\mathcal{C}$ and g out-propagates $\mathcal{C}$, then:*

$$(f \circ g)^{\mathcal{C}} \;\equiv\; f^{\mathcal{C}} \circ g^{\mathcal{C}} \;\; .$$

Combining Propositions 2.2–2.6, gives us the following:

Proposition 2.7 *If operators $f_1, \ldots, f_n$ are such that for some i, $0 \leq i \leq n$, f_j in-propagates $\mathcal{C}$ for each j such that $1 \leq j \leq i$, and f_j out-propagates $\mathcal{C}$ for each j such that $i < j \leq n$, then for every $x \in \mathcal{C}$:*

$$(f_1 \circ \ldots \circ f_n)(x) \;\subseteq\; x \quad \textit{iff} \quad (f_1^{\mathcal{C}} \circ \ldots \circ f_n^{\mathcal{C}})(x) \;\subseteq\; x \;\; . \tag{6}$$

By Proposition 2.7, if we can show that the image of a set of states can be computed by a sequence of out-propagating operations, followed by a sequence of in-propagating operations, then we can verify whether a given set is an invariant using approximate rather than exact operators.

It should be noted, that the result akin to Proposition 2.4–2.6 does not hold for the remaining combination of in- and out-propagating operators, i.e. if f out-propagates $\mathcal{C}$ and g in-propagates $\mathcal{C}$, then it might be the case that:

$$(f \circ g)^{\mathcal{C}} \not\equiv f^{\mathcal{C}} \circ g^{\mathcal{C}} \ .$$

This prevents us from devising a similar approach to verification by reachability analysis. In principle, if both the set of initial and the set of acceptable states belong to $\mathcal{C}$, we could try to compute the best approximation in $\mathcal{C}$ of reachable states, and then use Proposition 2.2 to complete the verification. Unfortunately, even if the image computation satisfies conditions from Proposition 2.7, unless it involves only in-propagating or only out-propagating operations, we cannot compute the best approximation of reachable states by replacing operators with their approximations. This is true because computing reachable states requires repeated image computation, which implies that some in-propagating operation must be followed by some out-propagating operation, in which case the best approximation property may be lost.

2.1 Proofs of key propositions

We give only the proofs of Propositions 2.4–2.7, since the proofs of Propositions 2.1–2.3 are straightforward.

Proof of Proposition 2.4 We first prove (4). If $x \in \mathcal{C}$, then:

$$\begin{aligned}(f \circ g)^{\mathcal{C}}(x) &= \lceil f(g(x)) \rceil^{\mathcal{C}} \\ &= f^{\mathcal{C}}(\lceil g(x) \rceil^{\mathcal{C}}) \\ &= f^{\mathcal{C}}(g^{\mathcal{C}}(\lceil x \rceil^{\mathcal{C}})) \\ &= f^{\mathcal{C}}(g^{\mathcal{C}}(x)) \\ &= (f^{\mathcal{C}} \circ g^{\mathcal{C}})(x) \ ,\end{aligned}$$

where the first equality holds by definition, the second by the in-propagation property of f, the third by the in-propagation property of g, the fourth follows from the fact that the assumption $x \in \mathcal{C}$ implies $x = \lceil x \rceil^{\mathcal{C}}$, and the final one by definition.

Now, we prove that $f \circ g$ in-propagates $\mathcal{C}$. Indeed, for any $x \in 2^S$:

$$\begin{aligned}\lceil (f \circ g)(x) \rceil^{\mathcal{C}} &= \lceil f(g(x)) \rceil^{\mathcal{C}} \\ &= f^{\mathcal{C}}(\lceil g(x) \rceil^{\mathcal{C}}) \\ &= f^{\mathcal{C}}(g^{\mathcal{C}}(\lceil x \rceil^{\mathcal{C}})) \\ &= (f^{\mathcal{C}} \circ g^{\mathcal{C}})(\lceil x \rceil^{\mathcal{C}}) \\ &= (f \circ g)^{\mathcal{C}}(\lceil x \rceil^{\mathcal{C}}) \ ,\end{aligned}$$

where the first equality holds by definition, the second by the in-propagation property of f, the third by the in-propagation property of g, the fourth by definition, and the final one by (4). □

Lemma 2.8 *If g out-propagates $\mathcal{C}$, then for every $x \in \mathcal{C}$:*

$$\begin{aligned}g(x) &= g(\lceil x \rceil^{\mathcal{C}}) \\ &= g^{\mathcal{C}}(\lceil x \rceil^{\mathcal{C}})\end{aligned}$$

$$= g^{\mathcal{C}}(x)$$
$$= \lceil g(x) \rceil^{\mathcal{C}} \ .$$

Proof. The first equality holds by the assumption that $x \in \mathcal{C}$, the second by the out-propagation property of g, the third again by $x \in \mathcal{C}$, and the forth by definition. □

Proof of Proposition 2.5 We first prove (5). If $x \in \mathcal{C}$, then:

$$\begin{aligned}(f \circ g)^{\mathcal{C}}(x) &= \lceil f(g(x)) \rceil^{\mathcal{C}} \\ &= \lceil f(g^{\mathcal{C}}(x)) \rceil^{\mathcal{C}} \\ &= f^{\mathcal{C}}(g^{\mathcal{C}}(x)) \\ &= (f^{\mathcal{C}} \circ g^{\mathcal{C}})(x) \ ,\end{aligned}$$

where the first equality holds by definition, the second by Lemma 2.8, the third by Lemma 2.8 (applied to f) and the final one by definition.

Now, we prove that $f \circ g$ out-propagates $\mathcal{C}$. Indeed, for any $x \in 2^S$:

$$\begin{aligned}(f \circ g)(\lceil x \rceil^{\mathcal{C}}) &= f(g(\lceil x \rceil^{\mathcal{C}})) \\ &= f(g^{\mathcal{C}}(\lceil x \rceil^{\mathcal{C}})) \\ &= f^{\mathcal{C}}(g^{\mathcal{C}}(\lceil x \rceil^{\mathcal{C}})) \\ &= (f^{\mathcal{C}} \circ g^{\mathcal{C}})(\lceil x \rceil^{\mathcal{C}}) \\ &= (f \circ g)^{\mathcal{C}}(\lceil x \rceil^{\mathcal{C}}) \ ,\end{aligned}$$

where the first equality holds by definition, the second by Lemma 2.8, the third by Lemma 2.8 (applied to f), the fourth by definition, and the final one by (5). □

Proof of Proposition 2.6 For any $x \in \mathcal{C}$:

$$\begin{aligned}(f \circ g)^{\mathcal{C}}(x) &= \lceil f(g(x)) \rceil^{\mathcal{C}} \\ &= f^{\mathcal{C}}(\lceil g(x) \rceil^{\mathcal{C}}) \\ &= f^{\mathcal{C}}(g^{\mathcal{C}}(x)) \\ &= (f^{\mathcal{C}} \circ g^{\mathcal{C}})(x) \ ,\end{aligned}$$

where the first equality holds by definition, the second the in-propagating property of f, the third by Lemma 2.8, and the final one by definition. □

Proof of Proposition 2.7 By Proposition 2.2 we have:

$$(f_1 \circ \ldots \circ f_n)(x) \ \subseteq \ x \quad \text{iff} \quad \lceil (f_1 \circ \ldots \circ f_n)(x) \rceil^{\mathcal{C}} \ \subseteq \ x \ , \tag{7}$$

which is equivalent to:

$$(f_1 \circ \ldots \circ f_n)(x) \ \subseteq \ x \quad \text{iff} \quad (f_1 \circ \ldots \circ f_n)^{\mathcal{C}}(x) \ \subseteq \ x \ , \ , \tag{8}$$

by the assumption $x \in \mathcal{C}$. From Proposition 2.6 and (8) it follows that:

$$\begin{aligned}&(f_1 \circ \ldots \circ f_n)(x) \ \subseteq \ x \quad \text{iff} \\ &\left((f_1 \circ \ldots \circ f_i)^{\mathcal{C}} \circ (f_{i+1} \circ \ldots \circ f_n)^{\mathcal{C}}\right)(x) \ \subseteq \ x\end{aligned} \tag{9}$$

because by Proposition 2.4 $f_1 \circ \ldots \circ f_i$ in-propagates $\mathcal{C}$, and by Proposition 2.5 $f_{i+1} \circ \ldots \circ f_n$ out-propagates $\mathcal{C}$. Finally, (9) implies (6), because

$$(f_1 \circ \ldots \circ f_i)^{\mathcal{C}} \equiv f_1^{\mathcal{C}} \circ \ldots \circ f_i^{\mathcal{C}}$$

by Proposition 2.4, and

$$(f_{i+1} \circ \ldots \circ f_n)^{\mathcal{C}} \equiv f_{i+1}^{\mathcal{C}} \circ \ldots \circ f_n^{\mathcal{C}}$$

by Proposition 2.5. □

3 Automata from Synchronous Languages

Compilers of synchronous languages such as Esterel [BC84] produce an automaton with states consisting of a finitely-valued component (called a *location*) and values of a finite number of integer variables. The transition relation is specified by a piece of sequential code associated with every location. This code contains neither loop nor recursion and it consists of three kinds of statements: *assignments* to integer variables, *tests* which select statements to be performed, and *branching* statements which terminate the code executed in a location and select the next location.

The approximate reachability analysis for such automata proposed in [Hal93] deals with the following special cases:

- An assignment may only increment, decrement, or reset to zero an integer variable. Such a variable is called a *counter*.
- The only tests are comparisons of counters with integers.

The proposed analysis is actually applicable to arbitrary automata generated by compilers of synchronous languages, but all other constructs are conservatively ignored. Automata satisfying these constraints are called *counter automata*. It is easy to see that their state space is infinite, and that the associate reachability problem is undecidable (because they subsume two-counter machines).

In the rest of this section we consider counter automata where counters are reals rather than integers. This does not change the behavior of these automata, but it could change the approximate analysis, because the constraint that counters can only be integers can be used to improve the approximation. However, this requires expensive integer linear programming. Therefore, we choose to follow the approach of the original approximate analysis [Hal93] where only real linear programming was used.

Given a set of states of a counter automaton, its image can be computed by executing symbolically the code specifying the transition relation. More precisely, to every statement we will assign a set of counter values as follows:

- To the first statement in the piece of code associated with some location, we associate the set counter values associated with that location in the given set of states.
- A statement is symbolically executed by transforming the set of counter values assigned to that statement (denoted by P), and assigning the transformed value(s) to its successor(s). The transformation rules are:
  - If the statement increments (decrements) the counter x, then we assign to its successor the set P translated along the x axis to the right (left) by one.

- If the statement resets x, then we assign to its successor the projection of P to the hyper-plane[3] $x = 0$.
- If the statement is of the form "**if** $x \leq c$ **then** $\{ A; \ldots \}$ **else** $\{ B; \ldots \}$", then we assign to A the intersection of P and half space $x \leq c$, and to B the intersection of P and half space $x > c$. Other inequality tests are analogous.
- If the statement is of the form "**if** $x = c$ **then** $\{ A; \ldots \}$ **else** $\{ B; \ldots \}$", then we assign to A the intersection of P and $x = c$, and to B the union of $P_>$ and $P_<$, where $P_>$ is the intersection of P and $x > c$, and $P_<$ is the intersection of P and $x < c$.

The image computation is completed by assigning to every location s the union of sets of counter values assigned to all instances of the statement **goto** s.

Let $\mathcal{C}_P$ denote the class containing all sets of states such that the set of counter values associated with some location form a polyhedron. It is not hard to check that the image of some set in $\mathcal{C}_P$ is also in $\mathcal{C}_P$. Since initial states of a counter automaton are in $\mathcal{C}_P$ (initially all the counters have value zero), it follows that throughout reachability analysis, the current set of reachable states is always in $\mathcal{C}_P$.

Halbwachs [Hal93] has proposed an approximate reachability algorithm based on using convex hulls to approximate many polyhedra. Since the convex hull of a polyhedron is always a *convex polyhedron*, it follows that approximations range over sets of states where the counter values associated with some location form a convex polyhedron. We use $\mathcal{C}_C$ to denote that class of approximations. Halbwachs have also proposed some additional approximation to ensure that the reachability analysis terminates . Since in invariant verification the termination is not an issue, we do not consider these additional approximations.

The class $\mathcal{C}_C$ is an example of an approximation class that is not closed under infinite intersection, and thus not all sets of states have the best approximation in $\mathcal{C}_C$. However, applying any operation in image computation to an element in $\mathcal{C}_C$ results in an element of $\mathcal{C}_P$, and every element of $\mathcal{C}_P$ does have the best approximation in $\mathcal{C}_C$.

Proposition 3.1 *Convex polyhedra are closed under intersection, translation, and projection.*

It follows that the symbolic execution of assignment statements, statements of the form "**if** $x \sim c \ldots$" where $\sim$ is some inequality operator, as well as the **then** branch of statements of the form "**if** $x = c \ldots$" all out-propagate $\mathcal{C}_C$. Other operations in the image computation (i.e. the **else** branch of statements of the form "**if** $x = c \ldots$" and the final union over branching statements) are addressed by the following result:

[3] Henceforth, we will use the term "projection" to mean "projection to a hyper-plane". Since these are the only projections we consider in this paper, this convention does not introduce any ambiguity.

Proposition 3.2 *Union, translation, and projection of polyhedra all in-propagate the set of convex polyhedra.*

Indeed, the convex hull of the union of two polyhedra is unchanged if they are replaced with their convex hulls, and computing the convex hull commutes with translation and projection.

To be able to combine these results with Proposition 2.7, we need to put some restrictions on the code associated with a location. We do not need any restrictions on assignment statements, because translation projection both in- and out-propagate convex polyhedra. However, we must ensure that after an operation which does not out-propagate $\mathcal{C}_C$ (i.e. union) we do not apply an operation which does not in-propagate $\mathcal{C}_C$ (i.e. intersection). We can ensure that by requiring that no **else** branch of a statement of the form "**if** $x = c \ldots$" contains any tests. If a counter automaton satisfies this condition, we say that it is *approximation ready*. For such automata, by applying Proposition 2.7, we have the following result about the utility of image approximation:

Proposition 3.3 *Let $\mathcal{T}$ denote the image operator associated with an approximation ready counter automata, and let $\mathcal{T}_c$ denote an approximation of $\mathcal{T}$ where all the intermediate results are approximated by their convex hulls. Then for every $x \in \mathcal{C}_C$:*

$$\mathcal{T}(x) \subseteq x \quad \textit{iff} \quad \mathcal{T}_c(x) \subseteq x \ .$$

Even if a counter automaton is not approximation ready, it is always possible to make it so by simple code transformations. For example, we can eliminate tests of the form $x = c$ by replacing every statement of the form:

if $x = c$ **then** A **else** B

with the following equivalent statement:

if $x < c$ **then** B **else** {**if** $x > c$ **then** B **else** A } .

Unfortunately, such transformations can increase the size of the code, and thus reduce the savings gained by computing an approximate image. However, if implemented carefully, the cost of associating a single polyhedron to a possibly larger number of statements should not be higher than the cost of associating a set of convex polyhedra (which is good representation of non-convex polyhedra) to fewer statements.

4 Timed Automata

Timed automata [AD90] are proposed as a model of real-time systems which extends finite-state automata with real-valued time-measuring devices called *timers*. Timers are used to bound the elapsed time between transitions. A timer can be reset to zero on any transition, and to any transition we can attach an enabling condition requiring that a timer has certain range of values.

Formally, if $x_1, \ldots, x_n$ are timers used in a timed automaton, then we say that a *timing constraint* is any expression of the form $x_i - x_j \sim c$ or $x_i \sim c$, where c is some integer and $\sim$ is $<, >, \leq$ or $\geq$. A *zone* is any convex

polyhedron representing solutions to a set of timing constraints. Given some integer c, we use Z_c to denote the set zones that can be specified using only timing constraints with the right hand side between 0 and c. It is easy to see that Z_c is finite for any c.

A state of a timed automaton is a pair (s, t) where s is one of the finitely many *locations*, and t is a real-valued vector such that component t_i represents the value of x_i. The transition relation of a timed automaton is specified by two functions:

- the *enabling function* E which assigns to every pair of locations a (possibly empty) zone representing enabling conditions for the corresponding transition, and
- the *reset function* R which assigns to every pair of locations a subset of timers that are to be reset on the corresponding transition.

Before we define the image operation, we need to introduce some notation. Given some vector t and a real number δ, we use $t + \delta$ to denote the vector which components satisfy $(t + \delta)_i = t_i + \delta$. Also, given some subset of timers X we use $t \downarrow_X$ to denote a vector which components satisfy $(t \downarrow_X)_i = 0$ if $x_i \in X$, and $(t \downarrow_X)_i = t_i$ otherwise.

The image of a set of states Q of a timed automaton can be computed by the following sequence of computation:

$$Q_1 = \{(s, t + \delta) \mid \delta \geq 0,\ (s, t) \in Q\} \ , \tag{10}$$

$$Q_2 = \{(s, s', t) \mid (s, t) \in Q_1,\ t \in E(s, s')\} \ , \tag{11}$$

$$Q_3 = \{(s, s', t \downarrow_{R(s,s')}) \mid (s, s', t) \in Q_2\} \ , \tag{12}$$

$$\mathcal{T}(Q) = \{(s', t) \mid \exists s \ : \ (s, s', t) \in Q_3\} \ . \tag{13}$$

The set Q_1 contains states reachable from Q by elapsing time, Q_2 contains all transitions that are enabled in some state in Q_1, Q_3 is the same as Q_2 except that the timers specified by R are reset, and finally $\mathcal{T}(Q)$ is obtained by removing the source location component from transitions in Q_3. The set $\mathcal{T}(Q)$ contains exactly the image of Q.

Alur and Dill have shown [AD90] that if the set of timer values associated with a location is a finite union of zones than so is its image. Since initially all timers have value zero (which is a zone), it follows that the reachability analysis can be performed by computing images of such sets only. This can be done effectively, because operations (10)–(13) extend to zones as follows:

- elapsing time (10) requires removing from the zone representation all upper-bound constraints on individual timers,
- restriction to enabled transitions (11) requires computing intersection of zones,
- reseting (12) requires computing projection of zones,
- existential quantification (13) requires computing union of zones.

Alur and Dill have also shown that if c is such that the range of E is contained in Z_c, then any zone appearing in the computation is also in Z_c. Since the range of E is finite, such a c always exists. It follows that only finitely

many zones may appear in the computation, and thus the reachability analysis always terminates. However, the number of zones may be very large, and in practice exact reachability analysis can be performed only on very simple timed automata.

4.1 Approximation by zones

To speed-up reachability analysis of timed automata, Balarin [Bal96] has proposed an approximate analysis where the union of two zones is approximated by the smallest zone containing both of them.[4] Throughout the proposed analysis, the set of timer values associated with some location is always a *single* zone in Z_c, for some integer c (that can easily be determined for a given timed automaton). We use $\mathcal{C}_{Z_c}$ to denote the class of such sets of states.

Proposition 4.1 *For any integer c, the set of zones Z_c is closed under removal of constraints, intersection, and projection.*

It follows that operations (10)–(12) all out-propagate $\mathcal{C}_{Z_c}$. In addition, since Z_c is finite and closed under intersection, it follows that the best approximation in Z_c (and thus also $\mathcal{C}_{Z_c}$) is well defined for any set of states.

Proposition 4.2 *For any integer c, union over sets of timer values in-propagates Z_c.*

It follows that operation (13) in-propagates $\mathcal{C}_{Z_c}$. Thus, we can now invoke Proposition 2.7 to prove the following:

Proposition 4.3 *Let an integer c, and a timed automaton with enabling function E and reset function R, be such that the range of E is contained in Z_c. Let $\mathcal{T}$ denote the image operator specified with E and R, and let $\mathcal{T}_{Z_c}$ be its approximation where every set of timer values associated to some location in some intermediate result is approximated by the smallest zone containing it. Then for every $x \in \mathcal{C}_{Z_c}$:*

$$\mathcal{T}(x) \subseteq x \quad \textit{iff} \quad \mathcal{T}_{Z_c}(x) \subseteq x \ .$$

4.2 Approximation by ordering

To further improve efficiency of the reachability analysis, an even weaker approximation has been proposed [Bal96], where the set of timer values associated with a location is always a zone that can be specified using only timing constraints of the form $x_i < x_j$ and $x_i \leq x_j$. We use $Z_<$ to denote the set of all such zones, and use $\mathcal{C}_{Z_<}$ to denote the corresponding set of states. In essence, using $\mathcal{C}_{Z_<}$ for approximation amounts to abstracting timer values and retaining only information about their relative ordering.

The first interesting property of $Z_<$ is that elapsing time is an identity operation for its elements. Indeed, since all timers advance at the same rate,

[4] Note that the smallest enclosing zone may strictly contain a convex hull of two zones, which is not necessarily a zone.

elapsing time does not change their order. Thus, only operations (11)–(13) are required for image computation.

Proposition 4.4 *The set of zones $Z_<$ is closed under intersection.*

It follows that any set of states has the best approximation in $\mathcal{C}_{Z_<}$ (because $Z_<$ is also finite), and that the restriction operation (11) out-propagates $\mathcal{C}_{Z_<}$.

Proposition 4.5 *Union and projection over sets of timer values both in-propagate $Z_<$.*

The last two results are sufficient to invoke Proposition 2.7, but the straightforward invocation would require unnecessary restrictions on the transition relation. By definition, values of the enabling function are arbitrary zones, and to use the out-propagating property of intersection, we would need to restrict them to be in $Z_<$. Fortunately, we can avoid this restriction by using the following asymmetric in-propagation property of intersection:

Proposition 4.6 *If zones x and y are such that $x \in Z_<$, then:*

$$\lceil x \cap y \rceil^{Z_<} \;=\; x \cap \lceil y \rceil^{Z_<} \;.$$

Indeed, since x contains only ordering information, to find ordering information implied by x and y together we may first extract the ordering information implied by y, and then combine it with that implied by x. We can use this property to replace an arbitrary transition relation with an approximation where every value of the enabling function is replaced with its best approximation in $Z_<$. Now we have all the pieces required for the following result:

Proposition 4.7 *Let $\mathcal{T}$ denote the image operator of some timed automata, and let $\mathcal{T}_{Z_<}$ be its approximation where every set of timer values associated to some location in some operand or some intermediate result is approximated by by the smallest enclosing zone in $Z_<$. Then for every $x \in \mathcal{C}_{Z_<}$:*

$$\mathcal{T}(x) \;\subseteq\; x \quad \textit{iff} \quad \mathcal{T}_{Z_<}(x) \;\subseteq\; x \;.$$

5 Conclusions

Formal verification of safety properties is a computationally intensive task, even if a user simplifies it by suggesting an invariant of the system. To alleviate this problem we have suggested to use approximate image computations, and we have shown that even though the approximation computes a superset of the image, it can, in certain cases, be used to answer categorically the question whether the suggested invariant contains its image.

In particular, we have showed that the three approximate image computation methods proposed previously for approximate reachability analysis could be used for the exact verification of an invariant, provided that the suggested invariant satisfies some conditions. In course of doing it, we have established quite general sufficient conditions that the approximate image computation

and the suggested invariant need to satisfy in order to always reach a conclusive result of the verification process.

In practice, the value of these results depends on how easy is it to find an invariant satisfying the constraints of the approximation class. In our very limited experience, we were able to find systems (concretely, the Fischer's mutual exclusion protocol) where the invariant belongs to the most restrictive class of approximations $\mathcal{C}_{Z<}$. We have also found that even if the proposed methods are not directly applicable, it is sometimes possible to refine a model by adding redundant state variables, and then use those variables to specify an invariant which does satisfy required restrictions.

In the future, we expect that the established general framework can be used to prove similar results in different settings. Another line of future research is to establish less restrictive conditions on the approximate image computation and the suggested invariant.

References

[ACHH93] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho. Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors, *Hybrid Systems, Lyngby, Denmark, 10-12 Oct. 1991, Proceedings*, pages 209–229. Springer-Verlag, 1993.

[AD90] Rajeev Alur and David L. Dill. Automata for modelling real-time systems. In M.S. Paterson, editor, *ICALP'90 Automata, languages, and programming: 17th international colloquium*. Springer-Verlag, 1990. LNCS vol. 443.

[Bal96] Felice Balarin. Approximate reachability analysis of timed automata. In *Proccedings of the 17th Real-Time Systems Symposium*, December 1996.

[BC84] G. Berry and L. Cosserat. The ESTEREL synchronous programming language and its mathematical semantics. In S.D. Brookes, A.W. Roscoe, and G. Winskel, editors, *Seminar on Concurrency*, pages 369–448. Carnegie-Mellon Univeristy, 1984.

[Hal93] Nicholas Halbwachs. Delay analysis in synchronous programs. In Costas Courcoubetis, editor, *Computer Aided Verification: 5th International Conference, CAV'93, Elounda, Greece, June/July 1993, Proceedings*, pages 333–346. Springer-Verlag, 1993. LNCS vol. 697.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, languages and Computation*. Addison Wesley, 1979.

[Kur94] Robert P. Kurshan. *Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach*. Princeton University Press, 1994.